CO2 Emissions by Fuel –

Statistical Tests and Linear Regression

by Zilma Karlla Barbosa Bezerra

Higher Diploma in Science in Data Analytics for Business

Data Preparation and Machine Learning

Aldana Louzán

CCT College

Dublin, Ireland

27th May 2022

Abstract

Carbon dioxide (CO2) emissions are a global issue that impacts climate change. The emissions have increased constantly throughout the years with some decreasing points of historical world events, such as the pandemic.

This report aims to analyse a dataset about CO2 emissions and execute three major tasks: a hypothesis test, a correlation analysis between two variables, and a linear regression model with the same variables.

The report follows the Cross-Industry Standard Process for Data Mining (CRISP-DM) methodology, a process model that describes common approaches in data analytics. This methodology will help organise the 3 primary requirements of this report with all the necessary steps to achieve them in a logical sequence.

The sections of CRISP-DM are Business Understanding, Data Understanding, Data Preparation, Modelling, Evaluation and Deployment. In the Business Understanding section, a general explanation of the business – or subject – is presented. Data Understanding is the section where the data is explored and described, including the use of some visualisations. At the end of this phase, the hypothesis test will be executed. In the third part, Data Preparation, the dataset is prepared for modelling and this happens by executing data formatting, selection of variables and correlation analysis. Then, a linear regression model will be executed in the Modelling stage. Finally, the project will be evaluated and deployed, in the last stage.

Table of Contents

Business Understanding 4

Data Understanding 5

Hypothesis Test 9

Data Preparation 12

Modelling 17

Evaluation & Deployment 21

References 22

Appendix 24

Data dictionary 24

Table of Figures

Figure 1. Basic information summary for the dataset. 5
Figure 2. Tail of the dataset. 6
Figure 3. Checking for duplicates and missing values. 6
Figure 4. Removing duplicates and resetting the index according to new shape of the dataset. 7
Figure 5. 5-number summary of numerical variables. 7
Figure 6. Count of unique values of the categorical variable "Model" as an example. 8
Figure 7. Histograms for the numerical variables. 8
Figure 8. Boxplots for the numerical variables. 9
Figure 9. Z-test parameters on statsmodels. 11
Figure 10. Z-test execution and result. 11
Figure 11. Dropping the columns "Make", "Model", "Vehicle Class" and "Transmission". 12
Figure 12. Encoding the variable "Fuel Type". 12
Figure 13. Renaming the new columns of fuel types. 13
Figure 14. Joining the new fuel columns to the original dataset. 13
Figure 15. Heatmap of the dataset. 14
Figure 16. Pairplot of the dataset. 15
Figure 17. Pairplot showing the relation between "Engine Size(L)" and "CO2 Emissions". 16
Figure 18. Linear Regression with Numpy Polyfit. 17
Figure 19. Graph showing the Linear Regression model built with Numpy. 18
Figure 20. Linear Regression with Scikit-learn, without splitting the data into training and test sets. 19
Figure 21. Linear Regression with Scikit-learn, splitting the dataset into training and test sets. 19
Figure 22. Comparing prediction and accuracy results. 20

Business Understanding

Since 1940, global carbon dioxide (CO2) emissions from fossil fuels and industry have increased almost every year. From 2000, the increase was even more considerable. A few years do not follow the same pattern, though. Some major global events can cause emissions reduction. For instance, in 2009 the global recession caused CO2 emissions to fall by 1.5%, approximately 460 million metric tons. In 2020, this reduction was even more significant: with a lot of countries under lockdown due to the COVID-19 pandemic, transportation and industrial activities suffered a substantial reduction and the global CO2 emissions plunged by over 5% (Tiseo, 2021a).

The CO2 emissions produced by passenger cars, specifically, have been steadily rising over the past two decades. Those vehicles produced approximately three billion metric tons of carbon dioxide emissions worldwide in 2020, which represents almost 10% of the overall emissions in the year and a 6% reduction in comparison to 2019, as a result of the pandemic (Tiseo, 2021a).

The urgency to reduce CO2 emissions is widely recognised, as they are the chief driver of global climate change (Ritchie and Roser, 2020). Because of this importance, the topic was the chosen one for this work.

Data Understanding

The dataset for this study is called “CO2_Emissions_Canada” and it was found on the Kaggle website[1]. The file is a compiled version of 7 years of data originally retrieved from the Canada Government Official Open Data website[2] and it captures the details of how CO2 emissions by vehicles can vary according to the different features like makes and models of cars, different types of fuels, and in different circumstances like driving in the city vs. driving in a highway. Though the author of the compiled dataset does not mention the exact years that the data refers to, the file is very helpful since the original files by year do not follow a pattern for the categorical variables, which would require an extra effort in cleaning and integrating data.

There are 7385 rows and 12 columns in the dataset and its attributes are: “Make”, “Model”, “Vehicle Class”, “Engine Size(L)”, “Cylinders”, “Transmission”, “Fuel Type”, “Fuel Consumption City (L/100 km)”, “Fuel Consumption Hwy (L/100 km)”, “Fuel Consumption Comb (L/100 km)”, “Fuel Consumption Comb (mpg)”, “CO2 Emissions(g/km)”. It is easy to detect, from here, that “CO2 Emissions(g/km)” is the dependent variable for this dataset. All this information can be seen by calling the pandas function *info* in Python (Figure 1). This function prints a concise summary of a dataset, including its index dtype and columns, non-null values and memory usage. A Data Dictionary can be found in the Appendix.

```
In [4]: # Obtaining basic information about the the dataset

        df.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 7385 entries, 0 to 7384
        Data columns (total 12 columns):
         #   Column                            Non-Null Count  Dtype
        ---  ------                            --------------  -----
         0   Make                              7385 non-null   object
         1   Model                             7385 non-null   object
         2   Vehicle Class                     7385 non-null   object
         3   Engine Size(L)                    7385 non-null   float64
         4   Cylinders                         7385 non-null   int64
         5   Transmission                      7385 non-null   object
         6   Fuel Type                         7385 non-null   object
         7   Fuel Consumption City (L/100 km)  7385 non-null   float64
         8   Fuel Consumption Hwy (L/100 km)   7385 non-null   float64
         9   Fuel Consumption Comb (L/100 km)  7385 non-null   float64
         10  Fuel Consumption Comb (mpg)       7385 non-null   int64
         11  CO2 Emissions(g/km)               7385 non-null   int64
        dtypes: float64(4), int64(3), object(5)
        memory usage: 692.5+ KB
```

Figure 1. Basic information summary for the dataset.

[1] *https://www.kaggle.com/debajyotipodder/co2-emission-by-vehicles*
[2] *https://open.canada.ca/data/en/dataset/98f1a129-f628-4ce4-b24d-6f16bf24dd64#wb-auto-6*

In the sequence, it is possible to observe samples of the dataset by applying the Pandas functions *head* and *tail* (Figure 2).

```
In [6]: # Observing a sample of the dataset - tail

        df.tail(10)
```

Out[6]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)	Fuel Consumption Comb (mpg)	CO2 Emissions(g/km)
7375	VOLVO	S90 T6 AWD	MID-SIZE	2.0	4	AS8	Z	11.3	7.5	9.6	29	223
7376	VOLVO	V60 T5	STATION WAGON - SMALL	2.0	4	AS8	Z	10.5	7.1	8.9	32	208
7377	VOLVO	V60 T6 AWD	STATION WAGON - SMALL	2.0	4	AS8	Z	11.0	7.4	9.4	30	219
7378	VOLVO	V60 CC T5 AWD	STATION WAGON - SMALL	2.0	4	AS8	Z	10.8	7.7	9.4	30	220
7379	VOLVO	XC40 T4 AWD	SUV - SMALL	2.0	4	AS8	X	10.2	7.5	9.0	31	210
7380	VOLVO	XC40 T5 AWD	SUV - SMALL	2.0	4	AS8	Z	10.7	7.7	9.4	30	219
7381	VOLVO	XC60 T5 AWD	SUV - SMALL	2.0	4	AS8	Z	11.2	8.3	9.9	29	232
7382	VOLVO	XC60 T6 AWD	SUV - SMALL	2.0	4	AS8	Z	11.7	8.6	10.3	27	240
7383	VOLVO	XC90 T5 AWD	SUV - STANDARD	2.0	4	AS8	Z	11.2	8.3	9.9	29	232
7384	VOLVO	XC90 T6 AWD	SUV - STANDARD	2.0	4	AS8	Z	12.2	8.7	10.7	26	248

Figure 2. Tail of the dataset.

While checking for duplicates, 1103 duplicated rows were found and removed. No missing values were found, though.

```
In [7]: # Checking if there are any missing values

        df.isnull().values.any()
```

```
Out[7]: False
```

```
In [8]: # Checking for duplicates

        df.duplicated().sum()
        print('There are a total of ' + (str(df.duplicated().sum()) + ' duplicates in the dataset.'))
```

```
There are a total of 1103 duplicates in the dataset.
```

Figure 3. Checking for duplicates and missing values.

It is possible to see that the number of entries is now 6282 instead of the initial 7385. This is due to the removal of the duplicates. After that, the index was reset so it would follow the new shape, going from 0 to 6281 (Figure 4).

```
In [9]: # Removing duplicates

        df.drop_duplicates(inplace = True)
        df.duplicated().sum()
        print('There are a total of ' + (str(df.duplicated().sum()) + ' duplicates in the dataset.'))

        There are a total of 0 duplicates in the dataset.

In [10]: # Checking the new shape of the dataset

         df.shape

Out[10]: (6282, 12)

In [11]: # Reseting the index

         df = df.reset_index(drop = True)
         df.tail()
```

Out[11]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)	Fuel Consumption Comb (mpg)	CO2 Emissions(g/km)
6277	VOLVO	XC40 T5 AWD	SUV - SMALL	2.0	4	AS8	Z	10.7	7.7	9.4	30	219
6278	VOLVO	XC60 T5 AWD	SUV - SMALL	2.0	4	AS8	Z	11.2	8.3	9.9	29	232
6279	VOLVO	XC60 T6 AWD	SUV - SMALL	2.0	4	AS8	Z	11.7	8.6	10.3	27	240
6280	VOLVO	XC90 T5 AWD	SUV - STANDARD	2.0	4	AS8	Z	11.2	8.3	9.9	29	232
6281	VOLVO	XC90 T6 AWD	SUV - STANDARD	2.0	4	AS8	Z	12.2	8.7	10.7	26	248

Figure 4. Removing duplicates and resetting the index according to new shape of the dataset.

Exploring the statistical summary of the dataset, this shows the following values for each variable: count, mean, std, min, 25%, 50%, 75% and max. As this summary is only applied for numerical variables, all the categorical variables were counted for unique values next (Figure 6 is an example). The first item on each list represents the mode of the column, meaning the value that is more frequent on it.

```
In [12]: # Obtaining a statistical summary for the dataset

         df.describe()
```

Out[12]:

	Engine Size(L)	Cylinders	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)	Fuel Consumption Comb (mpg)	CO2 Emissions(g/km)
count	6282.000000	6282.000000	6282.000000	6282.000000	6282.000000	6282.000000	6282.000000
mean	3.161812	5.618911	12.610220	9.070583	11.017876	27.411016	251.157752
std	1.365201	1.846250	3.553066	2.278884	2.946876	7.245318	59.290426
min	0.900000	3.000000	4.200000	4.000000	4.100000	11.000000	96.000000
25%	2.000000	4.000000	10.100000	7.500000	8.900000	22.000000	208.000000
50%	3.000000	6.000000	12.100000	8.700000	10.600000	27.000000	246.000000
75%	3.700000	6.000000	14.700000	10.300000	12.700000	32.000000	289.000000
max	8.400000	16.000000	30.600000	20.600000	26.100000	69.000000	522.000000

Figure 5. 5-number summary of numerical variables.

```
In [14]: # Counting unique values for 'Model'

         df['Model'].value_counts()

Out[14]: F-150 FFV             32
         F-150 FFV 4X4         31
         MUSTANG               27
         FOCUS FFV             24
         F-150 4X4             20
                               ..
         LS 500                 1
         LS 500h                1
         NX 300 AWD F SPORT     1
         RX 350 L AWD           1
         XC40 T4 AWD            1
         Name: Model, Length: 2053, dtype: int64
```

Figure 6. Count of unique values of the categorical variable "Model" as an example.

Histograms and boxplots were plotted, next, to check the distribution and look for outliers in the numerical variables (Figures 7 and 8, respectively).

Through the histogram for the variable “CO2 Emissions(g/km)”, it is possible to say that its distribution is close to a normal shape, though it is slightly positively skewed. The more bins were added to it, the closer to normal it looked.

```
In [18]: # Observing the distribution of the numerical variables

         df.plot(kind = 'hist', subplots = True, layout = (4, 3), figsize = (15, 15), bins = 100);
```

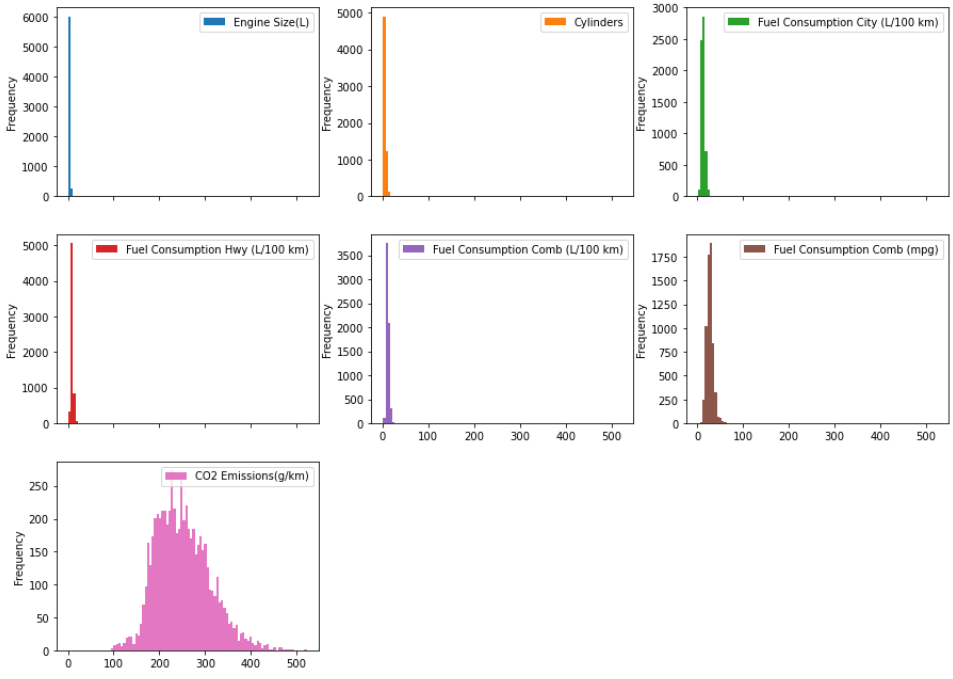


Figure 7. Histograms for the numerical variables.

A normal distribution implies that it is symmetrical about its central peak and it appears to have the right proportions as in the bell shape. Norm brings the sense of a standard, meaning idealised or perfect. Though the normal curve is a mathematical

abstraction, the bigger the population the closer it will be to the ideal (Rowntree, 2018, p. 69).

Skew is a long tail of observations. In this case, it is possible to observe a small tail towards the right side of the graph for “CO2 Emissions(g/km)”, which means a positive skew (Rowntree, 2018, p. 59).

Outliers are “sample values that lie very far away from the vast majority of the other sample values” (Triola, 2012, p. 46). In the boxplot, they are represented by the dots. It is possible to see that all variables have the presence of outliers. However, those will not be removed or replaced, as they can be important to detect if the cars are producing more or less CO2.

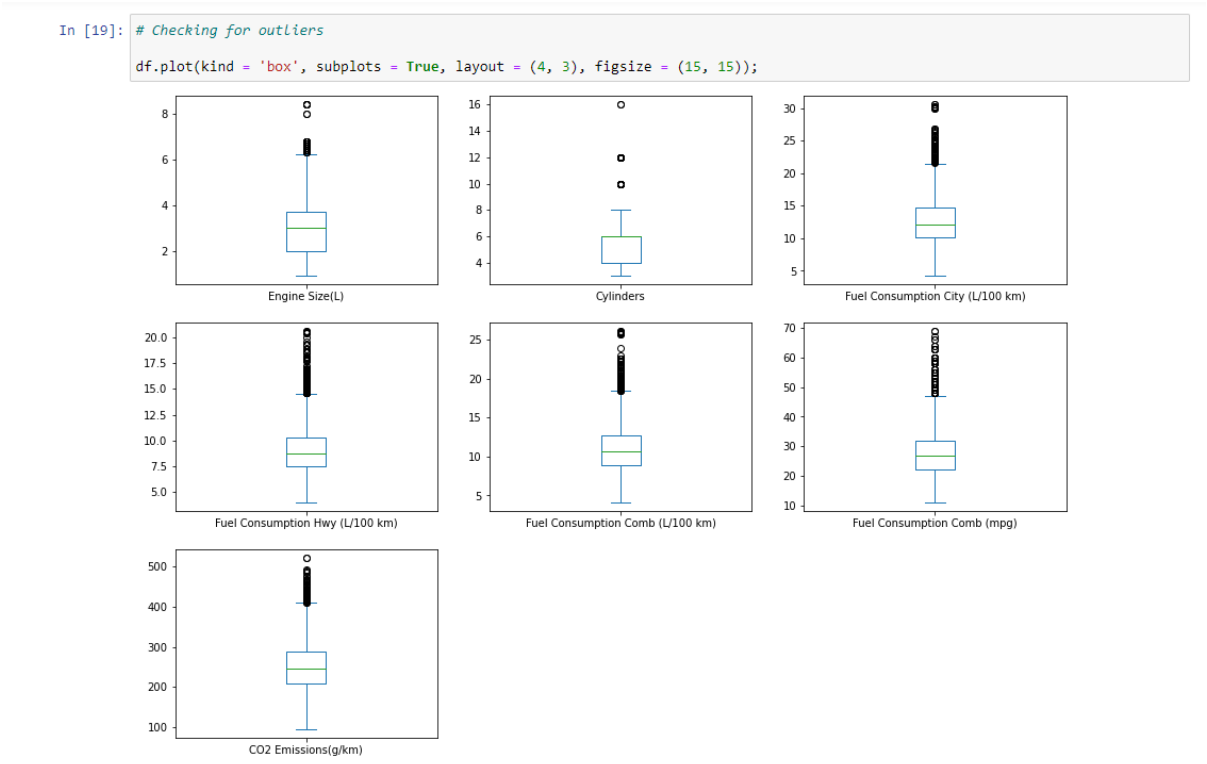


Figure 8. Boxplots for the numerical variables.

Hypothesis Test

As previously mentioned, the feature “CO2 Emissions(g/km)” was identified as the dependent variable in this dataset and it was chosen for the execution of a

hypothesis test. “The idea of hypothesis testing is to test whether we believe something to be true by looking at a sample of data” (Lakin, 2011, p. 195).

In order to execute hypothesis testing, two hypotheses are needed. The null hypothesis indicates no change or no difference, i.e., it claims that the value of some sample statistic is equal to the respective value of a population. On the other hand, the alternative hypothesis claims that the value being tested somehow differs from the null hypothesis (Triola, 2012, p. 395).

Other important concepts in hypothesis testing are population and sample. A population is a collection of all individuals to be studied and a sample is a subcollection of members of the population (Triola, 2012, p. 4). While sample figures are called statistics, population figures are called parameters and, through the statistics of the sample, it is possible to infer parameters for the population (Rowntree, 2018, pp. 82 and 83).

One of the many statistical tests available is the z-test. The letter z is commonly associated with normal distribution in statistics and is often used to test the properties of means (Lakin, 2011, p.202).

The requirements for a z-test are: the sample is random, the value of the population standard deviation is known and the population is normally distributed or the sample has more than 30 individuals (Triola, 2012, p. 425).

According to the United States Environmental Protection Agency (US EPA, 2018), the average passenger vehicle emits about 404 grams of CO2 per mile. That is the mean of the population. Converting the distance to km, which is the measure used in the dataset – the sample to be tested, the average passenger vehicle emits about 251 grams of CO2 per km.

So the null hypothesis is that the average passenger vehicle emits 251 grams of CO2 per km. The alternative hypothesis, on the other hand, is that the average passenger vehicle emits more than 251 grams of CO2 per km, as the mean of the sample is 251.15 g/km, as seen in the 5-number summary previously. So this will be a right-tailed test.

Though the standard deviation of the population is not available, it is assumed to be the same as the sample by the parameter *usevar* of the function *ztest* of the

statsmodels library. If *pooled*, then the standard deviation of the samples is assumed to be the same. Furthermore, *pooled* is the only available option currently (Perktold, Seabold and Taylor, 2019).

To run the ztest through the statsmodel, the following information is required: *x1* is the sample; *x2* is only used in the case of two samples being tested, *value* is the mean of the population – or the mean of the sample under the null hypothesis; *alternative* indicates if the alternative hypothesis is smaller, larger or different; *usevar*, as already explained, assumes that the standard deviation of the sample is the same; and *ddof* is set as 1 by default, meaning the degrees of freedom for comparison of means is one (Figure 9).

Figure 9. Z-test parameters on statsmodels.

The code returns the value for the z-test and the p-value, one of the methods of decision criteria used to reject or fail to reject the null hypothesis. The p-value "is the probability of getting a value of the test statistic that is at least as extreme as the one representing the sample data, assuming that the null hypothesis is true". If the p-value returned is less than 0.05, the null hypothesis is rejected and the alternative is accepted (Triola, 2012, pp. 400 and 402).

In this case, there was no sufficient evidence to reject the null hypothesis at the p-value 5%. Thus, it was accepted, meaning the average emissions of CO2 by a passenger vehicle in Canada is 251 g/km (Figure 10).

```
In [20]: # Hypothesis Test

         ## H0 = The average passenger vehicle in Canada emits 251g/km of CO2 => μ = 251
         ## H1 = The average passenger vehicle in Canada emits more the 251g/km of CO2 => μ > 251

         ztest, pval = stests.ztest(df['CO2 Emissions(g/km)'], x2 = None, value = 251, alternative = 'larger')
         print(float(ztest))
         print(float(pval))

         if pval < 0.10: # Alpha value is 0.05 or 5%
             print('Reject the null hypothesis.')
         else:
             print('Accept the null hypothesis.')

         0.2108822926538648
         0.41648956091534317
         Accept the null hypothesis.
```

Figure 10. Z-test execution and result.

Data Preparation

As the categorical columns “Make”, “Model”, “Vehicle Class” and “Transmission” have numerous unique observations, encoding them would make the dataset extremely large in the number of columns. So, they were dropped (Figure 11).

The only column that will be encoded is “Fuel Type”, so it will be possible to include it in the correlation analysis. The OneHotEncoder was the chosen method because it derives the categories based on the unique values of the feature and does not create any unwanted rank, as it would be with the OrdinalEncoder. The “Fuel Type” variable has only 5 unique observations, but one of them appears just once, so it was dropped. It is the letter N, that represents the natural gas. After that, the index was reset.

```
In [21]: # Dropping all the categorical variables, except 'Fuel Type'

         df = df.drop(columns = ['Make', 'Model', 'Vehicle Class', 'Transmission'], axis = 1)

In [22]: # Dropping the only observation with 'Fuel Type' = N

         df = df[df['Fuel Type'] != 'N']
         df = df.reset_index(drop = True)
         df.shape

Out[22]: (6281, 8)
```

Figure 11. Dropping the columns "Make", "Model", "Vehicle Class" and "Transmission".

The “Fuel Type” column is, then, encoded as a new dataset and, following, its columns are renamed accordingly (Figure 12).

```
In [23]: # Transforming the categorical column in an list of array

         fuel_type = df['Fuel Type'].array
         fuel_type = fuel_type.reshape(-1, 1)

In [24]: # Enconding the variable 'Fuel Type'

         ohe = OneHotEncoder(sparse = False)
         df_fuel_type = ohe.fit_transform(fuel_type)
         df_fuel_type = pd.DataFrame(df_fuel_type)
         df_fuel_type.head(50)

Out[24]:
```

	0	1	2	3
0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	1.0
2	0.0	0.0	0.0	1.0
3	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	1.0

Figure 12. Encoding the variable "Fuel Type".

```
In [26]: # Renaming the new columns of the encoded dataset

         df_fuel_type.columns = ['Fuel_D', 'Fuel_E', 'Fuel_X', 'Fuel_Z']
         df_fuel_type
```

Out[26]:

	Fuel_D	Fuel_E	Fuel_X	Fuel_Z
0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	1.0
2	0.0	0.0	0.0	1.0
3	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	1.0

Figure 13. Renaming the new columns of fuel types.

Finally, this new dataset is joined to the original one and the original column “Fuel Type” is dropped (Figure 14).

```
In [27]: # Joining the dataframes in one

         df = df.join(df_fuel_type)
         df = df.drop(columns = 'Fuel Type', axis = 1)
         df
```

Out[27]:

	Engine Size(L)	Cylinders	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)	Fuel Consumption Comb (mpg)	CO2 Emissions(g/km)	Fuel_D	Fuel_E	Fuel_X	Fuel_Z
0	2.0	4	9.9	6.7	8.5	33	196	0.0	0.0	0.0	1.0
1	2.4	4	11.2	7.7	9.6	29	221	0.0	0.0	0.0	1.0
2	1.5	4	6.0	5.8	5.9	48	136	0.0	0.0	0.0	1.0
3	3.5	6	12.7	9.1	11.1	25	255	0.0	0.0	0.0	1.0
4	3.5	6	12.1	8.7	10.6	27	244	0.0	0.0	0.0	1.0

Figure 14. Joining the new fuel columns to the original dataset.

Correlation Analysis

Before analysing the correlation between 2 specific variables, a heatmap and a pairplot were plotted for the whole dataset, in order to observe the correlation between all the variables and choose 2 for further analysis.

In the heatmap (Figure 15), it can be seen that all the fuel types do not have a strong correlation with “CO2 Emissions(g/km)”. On the other hand, “Engine Size(L)”, “Cylinders” and all types of fuel consumption have a strong correlation with CO2 emissions, having only one of them a negative correlation (“Fuel Consumption Comb (mpg)”). It was plotted by using the Pearson correlation coefficient and the scale set in the range from -1 to 1. The Pearson correlation is a numerical way to measure the correlation of a sample of data and, in the range between -1 and 1, 1 represents

perfect positive correlation and -1 represents perfect negative correlation (Lakin, 2011, p. 84). Furthermore, “it is accepted that the square of the correlation coefficient tells us how much of the variation in one variable can be explained by variations in the other” (Rowntree, 2018, p.173).

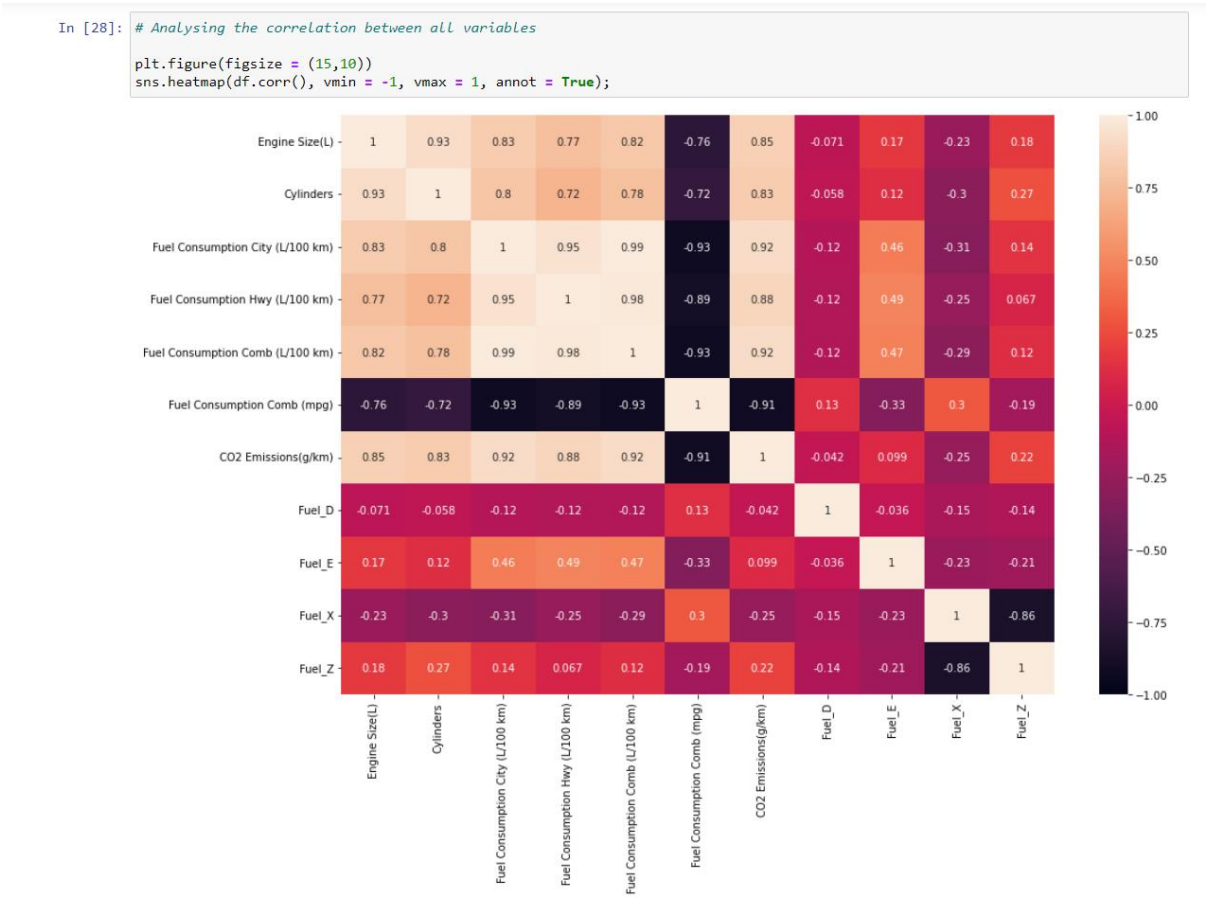


Figure 15. Heatmap of the dataset.

The strongest correlations with “CO2 Emissions(g/km)” mentioned above can be seen in the pairplot as linear (Figure 16). The relation between “Engine Size(L)” and “CO2 Emissions(g/km)” is a thicker line, more like a path. “Fuel Consumption City (L/100 km)”, “Fuel Consumption Hwy (L/100 km)” and “Fuel Consumption Comb (L/100 km)” follow the same pattern of two lines that are closer to the bottom of the graph and split as they move upwards. Finally, The strongest correlations with “CO2 Emissions(g/km)” mentioned above can be seen in the pairplot as linear. “Fuel Consumption Comb (mpg)” is the only one that moves downwards, as this is a negative correlation it is accepted that the square of the correlation coefficient tells

us how much of the variation in one variable can be explained by variations in the other, and has a curved shape.

```
In [29]: # Analysing the correlation between all variables - pairplot

all_pairs = sns.pairplot(df)

for ax in all_pairs.axes.flatten():
    ax.set_xlabel(ax.get_xlabel(), rotation = 45, fontsize = 16)
    ax.set_ylabel(ax.get_ylabel(), rotation = -45, fontsize = 16)
    ax.yaxis.get_label().set_horizontalalignment('right')
```

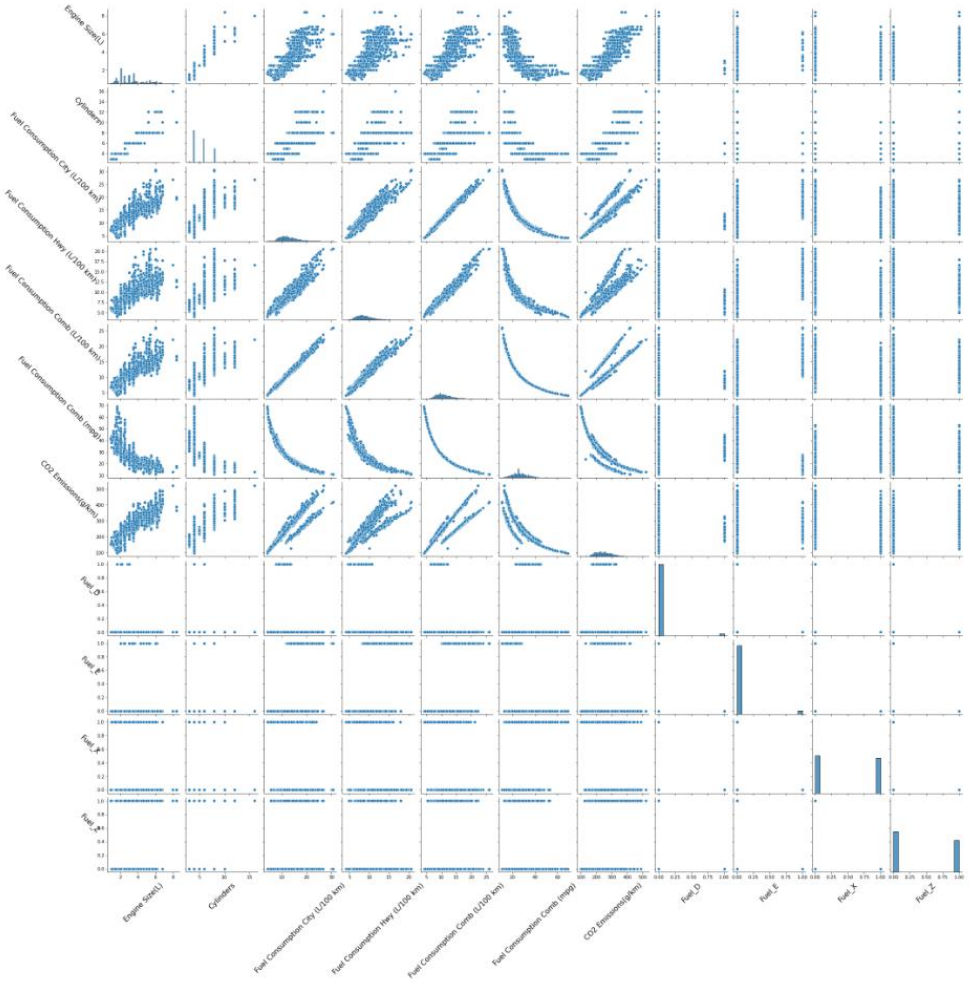


Figure 16. Pairplot of the dataset.

Although the strongest correlation with “CO2 Emissions(g/km)” is represented by “Fuel Consumption Comb (L/100 km)” and “Fuel Consumption City (L/100km)”, it is already known that fuel consumption influences the emissions of CO2. So, in this case, the variable that was chosen for further analysis was “Engine Size(L)”. The line is not well defined as the ones representing the strongest correlations, but the goal is to see if it is possible to predict CO2 emissions from the size of the engine of a car.

The ”Cylinders” variable was added as the parameter “hue”, only for visualisation purposes, as it is another interesting variable that follows a similar graph movement.

In the next section, a linear regression model will be built for those two variables.

```
In [30]: # Analysing the correlation between Engine Size and CO2 emissions


fuelcc_co2 = sns.pairplot(
    df,
    x_vars = ['Engine Size(L)'],
    y_vars = ['CO2 Emissions(g/km)'],
    hue = 'Cylinders',
    height = 7
)

fuelcc_co2.fig.suptitle("Engine Size x CO2 Emissions", y = 1.08, fontsize = 16);
```

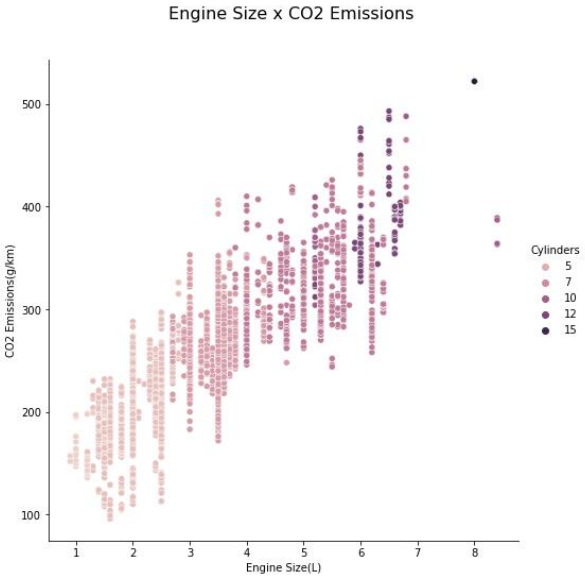


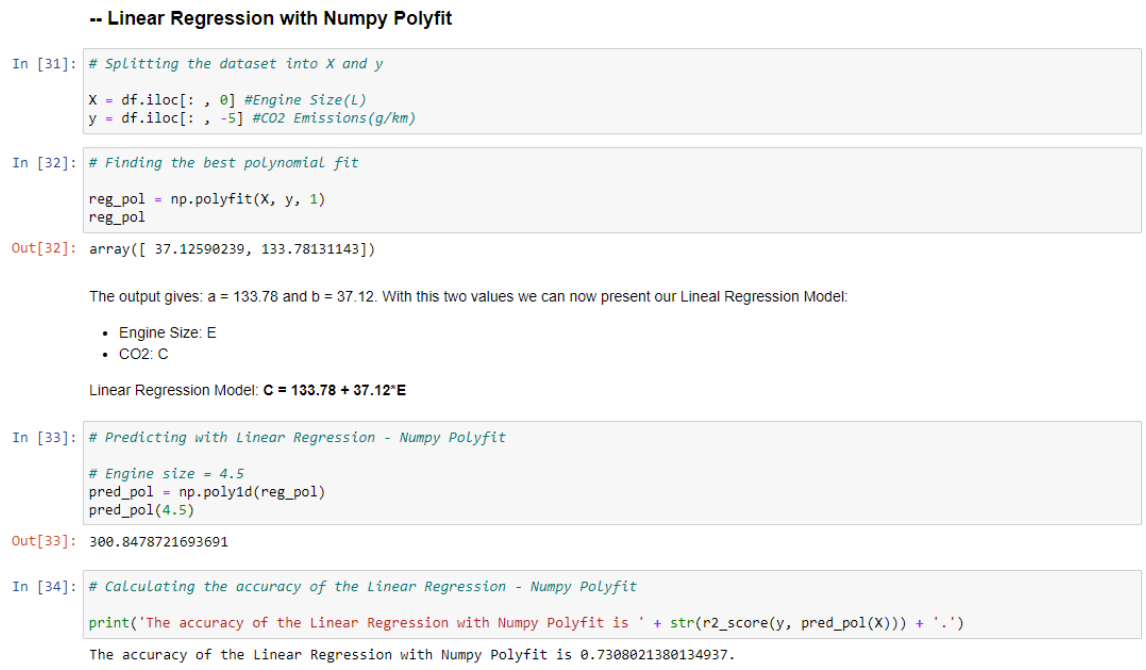
Figure 17. Pairplot showing the relation between "Engine Size(L)" and "CO2 Emissions".

Modelling

Linear regression “is the best straight line to draw that fits the data most accurately” (Lakin, 2011, p. 93). Since the model finds its best fit to the data, the line passes through the means of both variables, and this point is called the centroid.

With Numpy Polyfit it is possible to build a linear regression in Python. The function fits a polynomial to points of the variables chosen and returns a vector of coefficients that minimises the squared error. After fitting the model in the variables “Engine Size(L)” and “CO2 Emissions”, the values returned can be identified as the ones used in the linear regression formula. The first one represents the slope of the line (coefficient), while the second one represents the intercept (the value for y when X is zero).

With the model ready, it was possible to predict that an engine of size 4.5 L would emit approximately 300.85 g/km of CO2. The accuracy of this model is 73%. In the sequence, a graph was plotted showing how the line fits best the data.



-- Linear Regression with Numpy Polyfit

In [31]:
```
# Splitting the dataset into X and y

X = df.iloc[: , 0] #Engine Size(L)
y = df.iloc[: , -5] #CO2 Emissions(g/km)
```

In [32]:
```
# Finding the best polynomial fit

reg_pol = np.polyfit(X, y, 1)
reg_pol
```

Out[32]:
```
array([ 37.12590239, 133.78131143])
```

The output gives: a = 133.78 and b = 37.12. With this two values we can now present our Lineal Regression Model:

- Engine Size: E
- CO2: C

Linear Regression Model: **C = 133.78 + 37.12*E**

In [33]:
```
# Predicting with Linear Regression - Numpy Polyfit

# Engine size = 4.5
pred_pol = np.poly1d(reg_pol)
pred_pol(4.5)
```

Out[33]:
```
300.8478721693691
```

In [34]:
```
# Calculating the accuracy of the Linear Regression - Numpy Polyfit

print('The accuracy of the Linear Regression with Numpy Polyfit is ' + str(r2_score(y, pred_pol(X))) + '.')
```

```
The accuracy of the Linear Regression with Numpy Polyfit is 0.7308021380134937.
```

Figure 18. Linear Regression with Numpy Polyfit.

```
In [35]: # Plotting scatter plots and regression lines

         # Scatter plots
         ax1 = df.plot(kind = 'scatter', x = 'Engine Size(L)', y = 'CO2 Emissions(g/km)', color = 'blue', alpha = 0.5, figsize = (15, 10))

         # Regression lines
         plt.plot(X, reg_pol[0] * X + reg_pol[1], color = 'darkblue')

         # Regression equations
         plt.text(0, -25, 'y={:.2f}+{:.2f}*x'.format( reg_pol[1],  reg_pol[0]), color = 'darkblue')

         # legend, title and labels
         plt.legend(labels = ['Regresion Line', 'Engine Size(L)', 'CO2 Emissions(g/km)'])
         plt.title('Relationship between Engine Size and CO2 Emissions')
         plt.xlabel('Engine Size(L)')
         plt.ylabel('CO2 Emissions(g/km)');
```

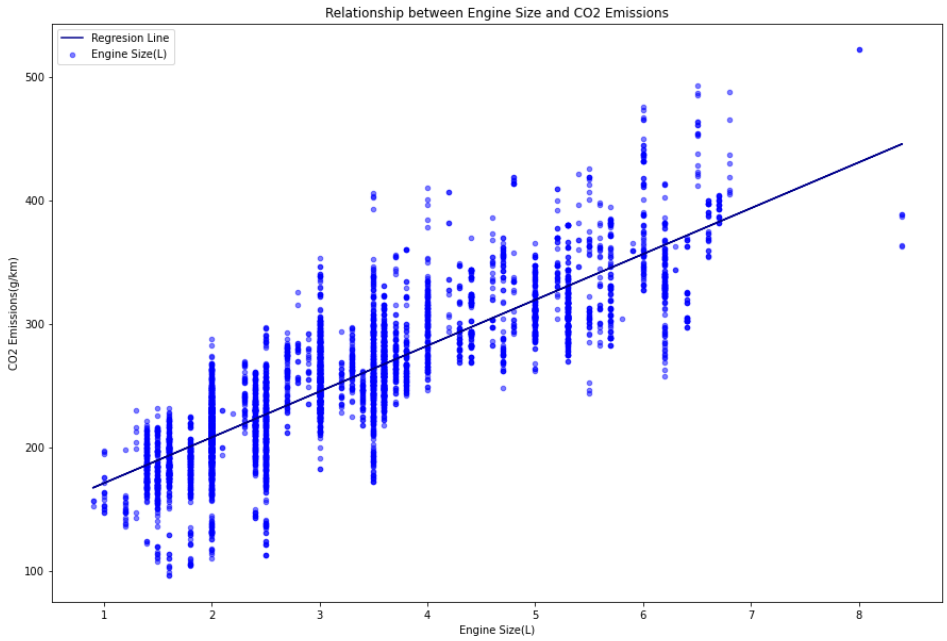


Figure 19. Graph showing the Linear Regression model built with Numpy.

The same model can be built with the *LinearRegression()* function from the Scikit-learn library and the result must be and it is the same. The same values for coefficient, intercept, prediction and intercept are found.

In order to take the experimentation one step further, the dataset was split into training and test sets and the model was applied in the training set, as it is usually done in real analysis. In this scenario, the values differ: both the predicted value and the accuracy score are slightly lower. This can be explained by the fact that the training set is a smaller sample than the whole dataset and the model gives a better result when it has more data to work on. All the values compared can be seen in Figure 22.

-- Linear Regression with Scikit-Learn - data not splitted into training and test set

In [36]:
```
# Creating linear regression object
reg_skl = LinearRegression()

# Fitting linear regression
reg_skl.fit(df[['Engine Size(L)']], df['CO2 Emissions(g/km)'])

# Getting the slope and intercept of the line best fit
print(reg_skl.intercept_)
print(reg_skl.coef_)
```

```
133.78131143234498
[37.12590239]
```

In [37]:
```
# Predicting the test set results for Linear Regression Scikit-Learn

y_pred = reg_skl.predict(X.array.reshape(-1, 1))
print(np.concatenate((y_pred.reshape(len(y_pred),1), y.values.reshape(len(y),1)),1))
```

```
[[208.0331162  196.        ]
 [222.88347716 221.        ]
 [189.47016501 136.        ]
 ...
 [208.0331162  240.        ]
 [208.0331162  232.        ]
 [208.0331162  248.        ]]
```

In [38]:
```
# Calculating the accuracy of the Linear Regression Scikit-Learn using the train / test split

print('The accuracy of the Linear Regression model, not using the train/test split is ' + str(reg_skl.score(X.array.reshape(-1, 1
```

```
The accuracy of the Linear Regression model, not using the train/test split is 0.7308021380134937.
```

Figure 20. Linear Regression with Scikit-learn, without splitting the data into training and test sets.

-- Linear Regression from Scikit-Learn - splitting the dataset intro training and test set

In [39]:
```
# Splitting the scaled dataset into the training set and test set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

In [40]:
```
# Training the Linear Regression Scikit-Learn on the training set

reg_skl_train = LinearRegression()
reg_skl_train.fit(X_train.array.reshape(-1, 1), y_train)
```

Out[40]:
```
LinearRegression()
```

In [41]:
```
# Predicting the test set results for Linear Regression Scikit-Learn

y_pred_train = reg_skl_train.predict(X_test.array.reshape(-1, 1))
print(np.concatenate((y_pred_train.reshape(len(y_pred_train),1), y_test.values.reshape(len(y_test),1)),1))
```

```
[[245.03039875 298.        ]
 [245.03039875 296.        ]
 [267.1861736  300.        ]
 ...
 [267.1861736  279.        ]
 [200.71884905 187.        ]
 [208.10410734 194.        ]]
```

In [42]:
```
# Calculating the accuracy of the Linear Regression Scikit-Learn using the train / test split

print('The accuracy of the Linear Regression model, using the train/test split is ' + str(reg_skl_train.score(X_train.array.resha
```

```
The accuracy of the Linear Regression model, using the train/test split is 0.7285316252877991.
```

Figure 21. Linear Regression with Scikit-learn, splitting the dataset into training and test sets.

-- Comparing predictions and accuracy in the three scenarios

In [43]:
```
# Comparing predictions of CO2 emissions for Engine Size = 4.5

# Polynomial Numpy
print(np.polyval(reg_pol, [4.5]))

# Linear Regression Scikit-Learn - without splitting the dataset into training and test set
print(reg_skl.predict([[4.5]]))

# Linear Regression Scikit-Learn - splitting the dataset into training and test set
print(reg_skl_train.predict([[4.5]]))
```

```
[300.84787217]
[300.84787217]
[300.41983588]
```

In [44]:
```
# Comparing accuracy of the linear regression

# Accuracy of the Linear Regression - Numpy Polyfit
acc_pol = r2_score(y, pred_pol(X))
print(acc_pol)

# Accuracy of the Linear Regression Scikit-Learn - data not splitted
acc_skl = reg_skl.score(X.array.reshape(-1, 1), y)
print(acc_skl)

# Accuracy of the Linear Regression Scikit-Learn using the train / test split
acc_skl_train = reg_skl_train.score(X_train.array.reshape(-1, 1), y_train)
print(acc_skl_train)
```

```
0.7308021380134937
0.7308021380134937
0.7285316252877991
```

Figure 22. Comparing prediction and accuracy results.

Evaluation & Deployment

In this report, a hypothesis test was conducted, with the null hypothesis being accepted and showing that passenger vehicles in Canada emit 251g of CO2 per kilometre.

A correlation analysis was carried out next and it was possible to observe that engine size, cylinders and fuel consumption rates are more correlated to CO2 emissions than the fuel type.

Finally, a linear regression model was built with Numpy and compared with the linear regression function available through the Scikit-learn library in two different scenarios. The model ran very well, with the same result for Numpy and Scikit-learn under the same circumstances (using the whole dataset), reaching a good accuracy score of 73%.

In a further analysis, in order to improve the accuracy, hyperparameters could be tested or the GridSearchCV could be applied to find the best parameters for the model. In addition, more variables could be used in multiple linear regression, as the combination of them can achieve better predictions.

Reference List

Government of Canada (2013). *Fuel Consumption Ratings - Open Government Portal*. [online] Government of Canada. Available at: https://open.canada.ca/data/en/dataset/98f1a129-f628-4ce4-b24d-6f16bf24dd64#wb-auto-6 [Accessed 23 May 2022].

Lakin, S. (2011). *How to Use Statistics*. Pearson Education Limited.

Matplotlib Development Team (2021). *Matplotlib: Python Plotting*. [online] Matplotlib.org. Available at: https://matplotlib.org/ [Accessed 23 May 2022].

NumPy developers (2022). *NumPy - the Fundamental Package for Scientific Computing with Python*. [online] Numpy.org. Available at: https://numpy.org/ [Accessed 23 May 2022].

Perktold, J., Seabold, S. and Taylor, J. (2019). *Statsmodels*. [online] www.statsmodels.org. Available at: https://www.statsmodels.org/dev/index.html [Accessed 26 May 2022].

Ritchie, H. and Roser, M. (2020). CO_2 and Greenhouse Gas Emissions. *Our World in Data*. [online] Available at: https://ourworldindata.org/co2-emissions#global-co2-emissions-from-fossil-fuels-global-co2-emissions-from-fossil-fuels [Accessed 23 May 2022].

Rowntree, D. (2018). *Statistics without Tears: an Introduction for Non-Mathematicians.* Penguin Books, Limited.

Scikit-learn developers (2022). *scikit-learn: machine learning in Python*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/ [Accessed 23 May 2022].

The pandas development team (2022). *pandas: Python Data Analysis Library*. [online] pandas. Available at: https://pandas.pydata.org/ [Accessed 19 May 2022].

The SciPy community (2020). *SciPy - Fundamental Algorithms for Scientific Computing in Python*. [online] Scipy.org. Available at: https://scipy.org/.

Tiseo, I. (2021a). *CO2 Emissions Worldwide*. [online] Statista. Available at: https://www.statista.com/statistics/276629/global-co2-emissions/ [Accessed 23 May 2022].

Tiseo, I. (2021b). *Global CO2 Emissions from Passenger Cars 2020*. [online] Statista. Available at: https://www.statista.com/statistics/1107970/carbon-dioxide-emissions-passenger-transport/#:~:text=Passenger%20cars%20produced%20approximately%20three [Accessed 23 May 2022].

Triola, M.F. (2012). *Elementary Statistics*. International Edition ed. Boston: Pearson Education, Inc.

US EPA (2018). *Greenhouse Gas Emissions from a Typical Passenger Vehicle*. [online] United States Environmental Protection Agency (US EPA). Available at: https://www.epa.gov/greenvehicles/greenhouse-gas-emissions-typical-passenger-vehicle [Accessed 25 May 2022].

Waskom, M. (2021). *seaborn: statistical data visualization — seaborn 0.10.1 documentation*. [online] seaborn.pydata.org. Available at: https://seaborn.pydata.org/index.html [Accessed 9 May 2022].

Appendix

Data dictionary

Col	Full Name of Variable	Definition of Variable	Type of Variable		Level of Measurement	Coding
			Qualitative / Quantitative	Discrete / Continuous		
A	Make	Make of the car	Qualitative	Discrete	Nominal	NA
B	Model	Model of the car	Qualitative	Discrete	Nominal	4WD/4X4 = Four-wheel drive AWD = All-wheel drive FFV = Flexible-fuel vehicle SWB = Short wheelbase LWB = Long wheelbase EWB = Extended wheelbase
C	Vehicle Class	Class of vehicle regarding its size	Qualitative	Discrete	Nominal	NA
D	Engine Size (L)	Size of engine in litres	Qualitative	Discrete	Ordinal	NA
E	Cylinders	Amount of cylinders	Qualitative	Discrete	Ordinal	NA
F	Transmission	Type of transmission	Qualitative	Discrete	Nominal	A = Automatic AM = Automated manual AS = Automatic with select shift AV = Continuously variable M = Manual 3 - 10 = Number of gears
G	Fuel Type	Type of fuel used by the vehicle	Qualitative	Discrete	Nominal	X = Regular gasoline Z = Premium gasoline D = Diesel E = Ethanol (E85) N = Natural gas
H	Fuel Consumption City (L/100 km)	Fuel consumption when driving in the city shown in litres per 100 kilometres (L/100 km)	Quantitative	Continuous	Ratio	NA
I	Fuel Consumption Hwy (L/100 km)	Fuel consumption when driving in a highway shown in litres per 100 kilometres (L/100 km)	Quantitative	Continuous	Ratio	NA
J	Fuel Consumption Comb (L/100 km)	Fuel consumption when driving in a combination of 55% in the city and 45% in a highway shown in litres per 100 kilometres (L/100 km)	Quantitative	Continuous	Ratio	NA
K	Fuel Consumption Comb (mpg)	Fuel consumption when driving in a combination of 55% in the city and 45% in a highway shown in miles per gallon (mpg)	Quantitative	Continuous	Ratio	NA
L	CO2 Emissions (g/km)	The tailpipe emissions of carbon dioxide for combined city and highway driving in grams per kilometre	Quantitative	Continuous	Ratio	NA